IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 10, 2020, accepted April 29, 2020, date of publication May 7, 2020, date of current version June 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2993103

GPU Acceleration of a Non-Standard Finite Element Mesh Truncation Technique for Electromagnetics

JOSÉ M. BADÍA[1], ADRIAN AMOR-MARTIN[2], (Member, IEEE), JOSE A. BELLOCH[3], (Member, IEEE), AND LUIS EMILIO GARCÍA-CASTILLO[4], (Member, IEEE)

[1]Departamento de Ingeniería y Ciencia de Computadores, Universitat Jaume I, 12071 Castellón, Spain
[2]Lehrstuhl für Theoretische Elektrotechnik, Universität des Saarlandes, 66123 Saarbrücken, Germany
[3]Departamento de Tecnología Electrónica, Universidad Carlos III de Madrid, 28911 Madrid, Spain
[4]Department of Signal Theory and Communications, Universidad Carlos III de Madrid, 28911 Madrid, Spain

Corresponding author: Adrian Amor-Martin (aamor@tsc.uc3m.es)

This work was supported in part by the Spanish Government under Grant TEC2016-80386-P, Grant TIN2017-82972-R, and Grant ESP2015-68245-C4-1-P, and in part by the Valencian Regional Government under Grant PROMETEO/2019/109.

ABSTRACT The emergence of General Purpose Graphics Processing Units (GPGPUs) provides new opportunities to accelerate applications involving a large number of regular computations. However, properly leveraging the computational resources of graphical processors is a very challenging task. In this paper, we use this kind of device to parallelize FE-IIEE (Finite Element-Iterative Integral Equation Evaluation), a non-standard finite element mesh truncation technique introduced by two of the authors. This application is computationally very demanding due to the amount, size and complexity of the data involved in the procedure. Besides, an efficient implementation becomes even more difficult if the parallelization has to maintain the complex workflow of the original code. The proposed implementation using CUDA applies different optimization techniques to improve performance. These include leveraging the fastest memories of the GPU and increasing the granularity of the computations to reduce the impact of memory access. We have applied our parallel algorithm to two real radiation and scattering problems demonstrating speedups higher than 140 on a state-of-the-art GPU.

INDEX TERMS CUDA, electromagnetics, finite elements, GPU.

I. INTRODUCTION

The Finite Element Method (FEM) is a well-known robust and versatile tool for the numerical solution of partial differential equations (PDEs) used in a wide variety of engineering disciplines and physics [1]–[4]. When FEM is used to model electromagnetic wave propagation in open region domains, e.g., in antenna analysis or in Radar Cross Section (RCS) prediction, the mesh generally needs to be truncated at some distance of the device/structure under analysis [5]. This kind of problems can also be found in other applications such as microwave tomographic imaging, [6], geophysics, [7], and nanotechnology, [8]. Since a volumetric mesh is needed, the number of finite elements (and hence unknowns of the problem) increases rapidly with this truncation distance, exerting a severe impact on the computational resources required to solve the problem. On the other hand, the mesh truncation technique also affects the accuracy of the FEM solution [5].

Different approaches for mesh truncation in electromagnetic wave propagation have been proposed in the literature. The use of the so-called ABC (Absorbing Boundary Conditions, see the review written in [9]) imposes locally the Sommerfeld radiation condition on the external (truncation) boundaries or the use of a Perfect Matched Layer (PML, [10]) which are natural choices for FEM. Although both numerical techniques are very different, they have in common the fact that they do not alter the sparse character of the matrices arising from FEM discretizations. However, both are approximate and they require a significant electrical distance in order to obtain accurate results. An alternative approach is to leverage a non-standard FEM discretization of the infinite exterior domain with infinite elements [11]. The use of infinite elements seems natural in the context of FEM but specific implementations are required and its adoption

The associate editor coordinating the review of this manuscript and approving it for publication was Chan Hwang See.

in electromagnetic wave propagation is not extended. The interested reader is referred to the literature for further details, e.g., [12]. On the other hand, an integral equation (IE) representation of the field in the infinite domain can be used (the so-called boundary elements [13]) on the interface between the exterior and interior domains. Boundary elements provide an exact radiation boundary condition at the continuous level, allowing the FEM domain to be truncated very close to (or even at zero distance from) the structure under test and, hence, reducing the number of unknowns of the problem. However, boundary elements provide a boundary condition that relates all unknowns associated with the truncation boundary obtaining dense matrices at the discrete level. The computation of the mentioned ''all-to-all'' relations involve costly convolutional operations using the Green's function (generally in free space) of the exterior domain.

Two of the authors have proposed a non-standard mesh truncation technique called FE-IIEE (Finite Element-Iterative Integral Equation Evaluation), which was introduced in the context of hybridization with asymptotic high frequency techniques (see [14]–[16]). FE-IIEE provides an asymptotically exact radiation boundary condition by using an IE representation of the exterior field while retaining the original sparse structure of the FEM algebraic system of equations. Thus, the truncation boundary can be placed very close to the structure under analysis without affecting the accuracy of the solution. The exterior scattered field is evaluated on a given boundary performing a small number of iterations which are computationally similar to post-process in the IE method. The coupling with FEM is performed by updating the residual of a local ABC on the exterior boundary, i.e., by simply updating the right-hand side (RHS) of the FEM algebraic system of equations.

The rest of the paper is structured as follows. In this section, we introduce the method to be parallelized, review related works, and depict our main goals. The FE-IIEE methodology and formulation are briefly described in Section II. Details about the GPU implementation of FE-IIEE are included in Section III. Section IV presents performance results. Finally, conclusions are drawn in Section V.

A. ACCELERATION OF THE FE-IIEE METHOD AND RELATED WORK

From the computational point of view, dense algebra operations and sparse matrix operations are clearly separated since the conventional flowchart of FEM is not altered and FE-IIEE appears as a post-process of the solution. Nevertheless, the convolutional character (double loop) associated to the computations leads to a computational complexity of $O(N^2)$, being N the number of unknowns associated to the exterior boundary. Thus, for very large problems, and depending on the computational resources available, it may be needed or convenient to introduce acceleration methods that yield to computational complexities of $O(N^\alpha)$ with $\alpha < 2$ at the expense of introducing some (error controlled) approximations. Several methods have appeared in the literature to accelerate these type of integral computations, such as the Fast Multipole Method (FMM) [17], [18], grid approaches based on Fast Fourier Transform (FFT) (e.g., [19]–[22]), and those based on algebraic compression (e.g., based on QR factorization [23], and Adaptive Cross Approximation (ACA), [24]). The application of ACA with FE-IIEE in the context of two-dimensional self-adaptive hp finite elements can be seen in [25].

The objective of this paper is to accelerate the computations involved in FE-IIEE by using Graphics Processing Units (GPUs). In the last years, GPUs have been a hot topic in the community of numerical methods, [26]–[29]. In the computational electromagnetics community specifically, a number of works have been reported, especially for integral equation techniques (Method of Moments —MoM—) and the Finite-Difference Time-Domain Method [30]–[34]. In [30], speedups of 30 in the MoM assembly and of 8 in the iterative solution with respect to a CPU are reported. CUDA and OpenMP are applied in [31], reporting a maximum speedup of 20 for the Multilevel Fast Multipole Algorithm (MLFMA). A cluster of GPUs and a similar algorithm is used in [32], showing a speedup of 82 with respect to a single CPU. An efficient strategy for the parallelization of the MLFMA is introduced in [34], using single-precision calculations to obtain a speedup of up to 98 with respect to the sequential version. In [33] a direct solver (LU decomposition) is applied to MoM obtaining a speedup of 38 relative also to the sequential version of the algorithm.

In terms of parallelization, FEM has a different nature since it is a communication-intensive problem: the workload of computing elemental matrices is small compared with the data communication needed for the finite element assembly and the matrix solution. Thus, in order to obtain reasonable performance, GPU-accelerated implementations typically need to introduce significant changes to the code workflow and data memory management. Good examples are [35]–[38] where special techniques to compute the elemental matrices for the finite element assembly are introduced and, specifically, iterative solvers are used increasing the speedup of the whole code [37], [38]. GPU acceleration has also been applied on the Discontinuous Galerkin Time Domain (DGTD) method, which shares some similarities with FEM. In this context, it is worth noting the work in [35] where speedups in the whole workflow of the code of up to 14 with respect to the sequential version are obtained. However, it must be emphasized that the use of discontinuous (non-conforming) approximations makes DGTD very different from typical FEM and much more suitable for parallelization.

As FE-IIEE is a non standard technique, its parallelization cannot be compared directly with any of the previously mentioned works. Note that, although FE-IIEE is used in the context of FEM as a mesh truncation technique, it uses an IE representation of the exterior field. In this sense, its parallelization is partially comparable with the parallelization of FEM+IE hybrid techniques. The work on GPU acceleration

presented in [39] is the most significant. In [39], GPU acceleration is introduced in all steps of the code workflow to obtain a speedup up to 24 (with multi-GPU implementation) with respect to a multi-threaded CPU solution. Nevertheless, there are important differences between [39] and the work presented in this paper. First, the target of this paper is only the FEM mesh truncation technique and not the whole FEM code. Because FE-IIEE is a domain decomposition method, it provides decoupling at the formulation level of the FEM part and the IE part, the latter appearing as a post-process of FEM in which the near scattered field is evaluated. The convolutional operation evaluating the near scattered field is also present in FEM+IE hybrid techniques, as [39], but it is later solved in the context of the evaluation of part of the coefficients of an algebraic system of equations (typically using an iterative solver and a preconditioner). In addition, the use of two separate surfaces as source and target in FE-IIEE, as it will be clear later, avoids the issue of the treatment of the singularities in the evaluation of the scattered field in conventional FEM+IE approaches.

B. OBJECTIVES USING GPU

Due to the intrinsic parallelism of FE-IIEE and the high computational cost of the floating point operations involved, FE-IIEE seems suitable to be accelerated using GPUs, and this is precisely the objective of the paper. However, the acceleration of the present FE-IIEE code implementation is still a challenging problem for this type of platform due to the amount, size and complexity of the data. It is important to point out that a self-imposed prerequisite in this work has been to perform the GPU acceleration of FE-IIEE without altering significantly the workflow and data structures of the original non-GPU code. It is well known that one of the main drawbacks when using GPU programming is the significant modification of the original code that usually undergoes in terms of algorithm workflow and data structures in order to adapt it to future GPU architectures and preserve the maintainability of the code. Thus, the challenge has to be understood also in this context.

In order to achieve the acceleration of present FE-IIEE implementation, several strategies are employed to leverage the manycore architecture of the GPU using CUDA [40]. In the CUDA model, the programmer defines the kernel function including the code to be executed on every core of the GPU. A grid configuration, which defines the number of threads and how they are distributed and grouped, must be built into the main code. The total number of threads running a kernel by means of thread blocks can exceed the number of physical cores. At runtime, the scheduler distributes the thread blocks among Streaming Multiprocessors (SMs). Therefore, most of the GPU-based research works in different fields, such as [41], [42], using a priori analysis of thread blocks and grids sizes to optimize computational efficiency. Other important issues to take into account are the occupancy of the GPU, the efficient use of the fastest memory levels of the GPU, and the granularity of the computations on every thread [43, Chap. 5]. By considering all of these factors, we will show considerable acceleration ratios while keeping the fundamental workflow of the initial sequential code.

II. FE-IIEE FORMULATION

The FEM code where FE-IIEE is introduced is based on a weak formulation with a double-curl vector wave equation in the frequency domain (i.e., time-harmonic case) for a given problem domain Ω^{FEM},

$$\boldsymbol{\nabla} \times \frac{1}{\mu_r}(\boldsymbol{\nabla} \times \boldsymbol{E}) - k_0^2 \varepsilon_r \boldsymbol{E} = \boldsymbol{O}, \quad \text{on } \Omega^{\text{FEM}} \tag{1}$$

where $\boldsymbol{E}$ is the electric field, k_0 is the wavenumber in vacuum, and $\boldsymbol{O}$ is the excitation term which is related to impressed currents within the problem domain. Note that the vector fields and currents are phasors represented by complex numbers, i.e., $\boldsymbol{v}(\boldsymbol{r}, t) = \boldsymbol{V}(\boldsymbol{r}) \exp(j\omega t)$, where $\boldsymbol{v}$ is a given vector field and $\boldsymbol{V}$ its phasor. Symbol ω stands for the angular frequency and j stands for the imaginary unit. The BVP (Boundary Value Problem) is closed through the imposition of Dirichlet, Neumann and Cauchy boundary conditions on Γ_{D}, Γ_{N} and Γ_{C} boundaries respectively,

$$\hat{\boldsymbol{n}} \times \boldsymbol{E} = 0, \quad \text{on } \Gamma_{\text{D}}, \tag{2}$$

$$\hat{\boldsymbol{n}} \times \frac{1}{\mu_r}(\boldsymbol{\nabla} \times \boldsymbol{E}) = 0, \quad \text{on } \Gamma_{\text{N}}, \tag{3}$$

$$\hat{\boldsymbol{n}} \times \frac{1}{\mu_r}(\boldsymbol{\nabla} \times \boldsymbol{E}) + jk_0(\hat{\boldsymbol{n}} \times \hat{\boldsymbol{n}} \times \boldsymbol{E}) = \boldsymbol{\Psi}, \quad \text{on } \Gamma_{\text{C}}, \tag{4}$$

where $\hat{\boldsymbol{n}}$ is the outward normal unit vector on the surface where the boundary condition is applied, and $\boldsymbol{\Psi}$ can be either the truncation boundary for open region problems (as it is the case with FE-IIEE) or the excitation related to a waveport. Note that a dual formulation can be defined with the magnetic field $\boldsymbol{H}$ as the unknown. For simplicity in the notation the formulation is restricted to the electric field $\boldsymbol{E}$.

Then, the variational formulation of the electromagnetic problem is found applying the Galerkin method on (1)-(4): Find $\boldsymbol{E} \in \boldsymbol{W}$ such that

$$c_1(\boldsymbol{F}, \boldsymbol{E}) - k_0^2\, c_2(\boldsymbol{F}, \boldsymbol{E}) + \gamma\, c_3(\boldsymbol{F}, \boldsymbol{E}) = l(\boldsymbol{F}), \quad \forall \boldsymbol{F} \in \boldsymbol{W}, \tag{5}$$

where the bilinear (c_1, c_2 and c_3) and linear forms (l, $l_{\boldsymbol{\Psi}}$) are defined as follows

$$c_1(\boldsymbol{F}, \boldsymbol{E}) = \int_{\Omega} (\boldsymbol{\nabla} \times \boldsymbol{F}) \cdot \left(\frac{1}{\mu_r} \boldsymbol{\nabla} \times \boldsymbol{E} \right) d\Omega,$$

$$c_2(\boldsymbol{F}, \boldsymbol{E}) = \int_{\Omega} \boldsymbol{F} \cdot \varepsilon_r \boldsymbol{E}\, d\Omega,$$

$$c_3(\boldsymbol{F}, \boldsymbol{E}) = \int_{\Gamma_{\text{C}}} \left(\hat{\boldsymbol{n}} \times \boldsymbol{F}\right) \cdot \left(\hat{\boldsymbol{n}} \times \boldsymbol{E}\right) d\Gamma_{\text{C}}, \tag{6}$$

$$l(\boldsymbol{F}) = \int_{\Omega} (\boldsymbol{F} \cdot \boldsymbol{O})\, d\Omega - l_{\boldsymbol{\Psi}}(\boldsymbol{F}), \tag{7}$$

$$l_{\boldsymbol{\Psi}}(\boldsymbol{F}) = \int_{\Gamma_{\text{C}}} (\boldsymbol{F} \cdot \boldsymbol{\Psi})\, d\Gamma_{\text{C}}, \tag{8}$$

while the space of functions is defined as, [1],

$$\boldsymbol{W} := \{\boldsymbol{A} \in \boldsymbol{H}(\text{curl}, \Omega),\ \hat{\boldsymbol{n}} \times \boldsymbol{A} = 0 \text{ on } \Gamma_D\}. \tag{9}$$

The discretization of the problem is realized through the tessellation of Ω^{FEM} with tetrahedra and the use of the second order isoparametric curl-conforming basis functions $\boldsymbol{N}_i$ ($\boldsymbol{E} = \sum_i g_i \boldsymbol{N}_i$) presented in [44]. As a result, a FEM algebraic system of equations $[A]\boldsymbol{g} = \boldsymbol{b}$ is produced, where the g_i defined above are gathered with $\boldsymbol{g}$.

The problem domain Ω^{FEM} needs to be truncated for open region problems. Specifically for FE-IIEE, (4) is employed on the truncation/exterior boundary (denoted as surface S hereon). The vector excitation $\boldsymbol{\Psi}$ is set to $\boldsymbol{0}$ for a radiation/antenna problem (given that the excitation is within S), while $\boldsymbol{\Psi} = \boldsymbol{\Psi}_{\text{inc}}$ for a scattering problem (in this case, the excitation is exterior to S, e.g., for RCS prediction). The vector function $\boldsymbol{\Psi}_{\text{inc}}$ is obtained by substitution of $\boldsymbol{E} = \boldsymbol{E}_{\text{inc}}$ in (4), where $\boldsymbol{E}_{\text{inc}}$ is the given incident field for the problem. Note that $\boldsymbol{\Psi}$ is only analytically known when S is at infinite distance from the sources (i.e., the structure under analysis). When S is at a finite distance from the sources, $\boldsymbol{\Psi}$ is not known and has to be numerically estimated. In particular, if S is close to the sources and $\boldsymbol{\Psi}$ is considered as if S would be at infinity, the error in $\boldsymbol{\Psi}$ affects severely the $\boldsymbol{E}$ field solution, [5]. In this context, FE-IIEE iteratively updates $\boldsymbol{\Psi}$ providing an arbitrarily accurate estimation and, hence, an asymptotically exact radiation boundary condition on S. The i-th update of $\boldsymbol{\Psi}$ is noted in the following as $\boldsymbol{\Psi}^{(i)}$. These iterations are seen by the FEM code as a post-process of the FEM solution performed independently from the main flow of the "conventional" FEM (see Fig. 1).

Note that FE-IIEE can be viewed as a non-standard multiplicative Schwarz domain decomposition method where the original infinite domain is divided into two overlapping domains: a finite FEM domain Ω^{FEM} bounded by S, and an infinite domain exterior to an auxiliary boundary S' which is located within S, as shown in Fig. 2. The overlapping region is the volume between surfaces S' and S and affects substantially to the convergence of the method. At each iteration of FE-IIEE, the scattered field $\boldsymbol{E}_{\text{FE-IIEE}}$ and its curl $\boldsymbol{\nabla} \times \boldsymbol{E}_{\text{FE-IIEE}}$ are computed on the surface S from the radiation of the equivalent electric and magnetic currents on an auxiliary surface S', $\boldsymbol{J}_{\text{eq}}, \boldsymbol{M}_{\text{eq}}$, respectively. This is realized by using the Equivalence Principle (see for instance [45, Chap. 3]). Thus, for $\boldsymbol{r} \in S$ and $\boldsymbol{r'} \in S'$, we compute

$$\begin{aligned}\boldsymbol{E}_{\text{FE-IIEE}}(\boldsymbol{r}) = &\oiint_{S'} (\boldsymbol{M}_{\text{eq}}(\boldsymbol{r'}) \times \boldsymbol{\nabla} G(\boldsymbol{r}, \boldsymbol{r'}) dS' \\ &- jk\eta \oiint_{S'} (\boldsymbol{J}_{\text{eq}}(\boldsymbol{r'})(G + \frac{1}{k^2} \boldsymbol{\nabla}\boldsymbol{\nabla} G(\boldsymbol{r}, \boldsymbol{r'}))) dS', \end{aligned} \tag{10}$$

$$\begin{aligned}\boldsymbol{\nabla} \times \boldsymbol{E}_{\text{FE-IIEE}}(\boldsymbol{r}) = &jk\eta \oiint_{S'} (\boldsymbol{J}_{\text{eq}}(\boldsymbol{r'}) \times \boldsymbol{\nabla} G(\boldsymbol{r}, \boldsymbol{r'})) dS' \\ &- \oiint_{S'} (\boldsymbol{M}_{\text{eq}}(\boldsymbol{r'})(k^2 G(\boldsymbol{r}, \boldsymbol{r'}) + \boldsymbol{\nabla}\boldsymbol{\nabla} G(\boldsymbol{r}, \boldsymbol{r'}))) dS', \end{aligned} \tag{11}$$

where k and η denote the wavenumber and characteristic impedance of the medium exterior to S', and G is the Green's function which, for a homogeneous exterior region, is

$$G(\boldsymbol{r}, \boldsymbol{r'}) = \frac{1}{4\pi R} e^{-jkR}, \tag{12}$$

where $R = \boldsymbol{r} - \boldsymbol{r'}$. By observing the above expressions, it is clear the convolutional character of the operations involved in the computation of $\boldsymbol{E}_{\text{FE-IIEE}}$ and its curl.

The electric and magnetic currents on S' used in (10), (11) are obtained as the tangential component of the numerical magnetic and electric field given by FEM, i.e.,

$$\begin{aligned}\boldsymbol{J}_{\text{eq}}(\boldsymbol{r'}) &= \hat{\boldsymbol{n}} \times \boldsymbol{H}_{\text{FEM}} = -\frac{1}{jk\eta} \hat{\boldsymbol{n}} \times \left(\sum_i g_i (\boldsymbol{\nabla} \times \boldsymbol{N}_i(\boldsymbol{r'})) \right), \\ \boldsymbol{M}_{\text{eq}}(\boldsymbol{r'}) &= -\hat{\boldsymbol{n}} \times \boldsymbol{E}_{\text{FEM}} = -\hat{\boldsymbol{n}} \times \left(\sum_i g_i \boldsymbol{N}_i(\boldsymbol{r'}) \right). \end{aligned} \tag{13}$$

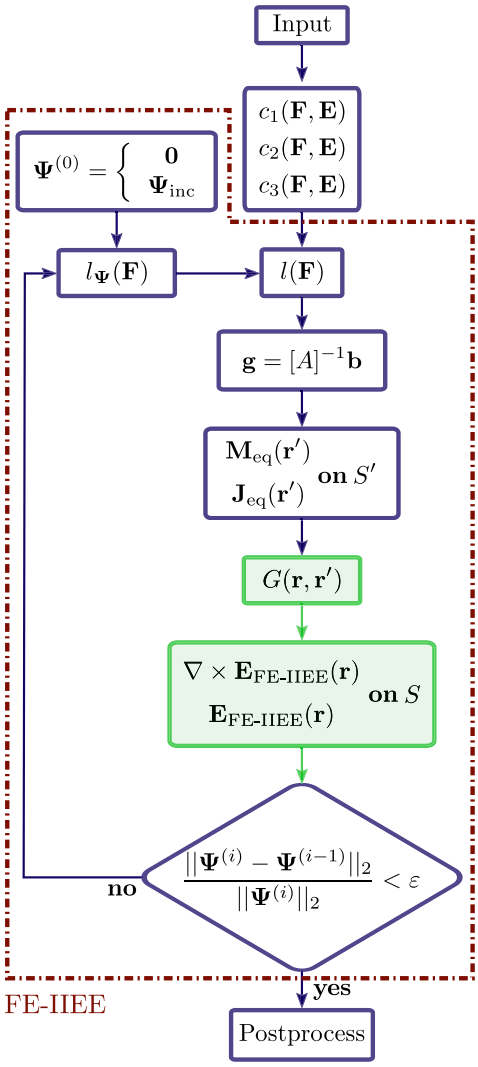

FIGURE 1. Workflow of the whole FEM code. The dashed line encompasses the operations related to the FE-IIEE method. Green boxes note GPU accelerated operations.

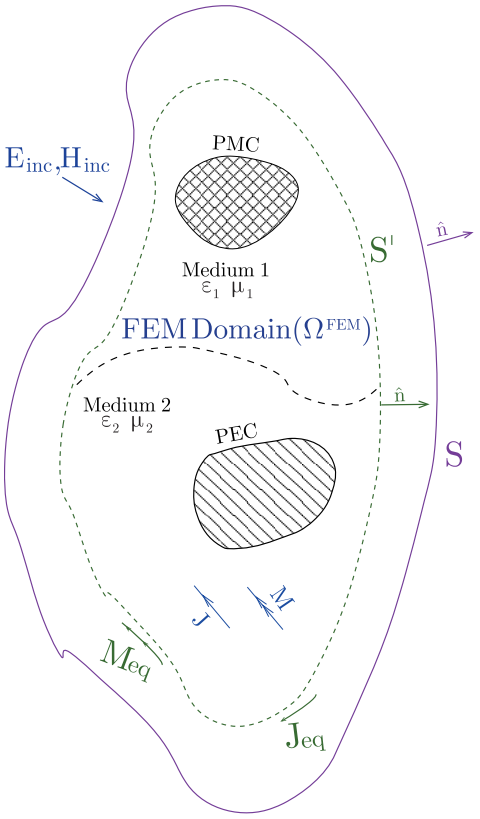

FIGURE 2. Decomposition in two overlapping domains for FE-IIEE.

Once $\boldsymbol{E}_{\text{FE-IIEE}}$ and its curl are computed, they are substituted into (4),

$$\boldsymbol{\Psi}^{(i)} = \boldsymbol{\Psi}_{\text{inc}} + \hat{\boldsymbol{n}} \times \frac{1}{\mu_r}(\boldsymbol{\nabla} \times \boldsymbol{E}_{\text{FE-IIEE}}) + jk_0(\hat{\boldsymbol{n}} \times \hat{\boldsymbol{n}} \times \boldsymbol{E}_{\text{FE-IIEE}}). \tag{14}$$

This $\boldsymbol{\Psi}^{(i)}$ is used to compute $l_{\boldsymbol{\Psi}}(\boldsymbol{F})$ using (8) which belongs to the RHS of the FEM system of equations.

FE-IIEE has already been successfully applied to a number of problems (see [14]–[16] and [12], [46]–[52]). In practice, the distance between S and S' is a small fraction of the wavelength, typically in the range of $0.05\,\lambda$ to $0.2\,\lambda$, hence allowing the truncation boundary to be placed very close to the sources. Provided S is convex, the convergence is monotonous and nicely smooth in all cases tested. The overlapping between the domains, i.e., the distance between S and S', affects the speed of convergence; i.e., the number of iterations needed to achieve a given error level. Thus, from the computational point of view, there is a compromise between the number of iterations and the size of the problem. The interested reader is referred to [46] for further details.

The workflow of the FE-IIEE algorithm is shown in Fig. 1. Here, the starting point is assumed to be after the FEM assembly, see (7). First, an initial guess is set for the first iteration of FE-IIEE. For antenna/radiation problems, $\boldsymbol{\Psi}^{(0)}$ is set to zero. For scattering problems $\boldsymbol{\Psi}^{(0)} = \boldsymbol{\Psi}_{\text{inc}}$. Then, the part of the RHS dependent on $\boldsymbol{\Psi}^i$, i.e., $l_{\boldsymbol{\Psi}}$ of (8), is computed. Next, the whole problem is solved. Note that if a direct sparse solver is used, the factorized matrix may be stored in order to compute only forward and backward substitutions (which are computationally cheap) when modifying the RHS in the subsequent iterations. To compute the value of $\boldsymbol{\Psi}^{(i)}$, (13), (12), (10), (11) and (14) are obtained consecutively. Finally, if the incremental error $\frac{||\boldsymbol{\Psi}^{(i)} - \boldsymbol{\Psi}^{(i-1)}||_2}{||\boldsymbol{\Psi}^{(i)}||_2}$ is lower than a given tolerance ε, the final solution has been computed and only remains to calculate the desired post-process. Otherwise, the number of iterations may also be fixed.

Note that the part of this workflow accelerated with GPU corresponds to the steps colored in green in Fig. 1. These steps concentrate most of the computational work of FE-IIEE without taking into account the obtention of the FEM solution system of equations. However, when the FEM matrix is factorized (as it is the case in this paper), only relatively cheap forward and backward substitutions are performed. The parallelization of the computation of $\boldsymbol{E}_{\text{FE-IIEE}}$ and $\boldsymbol{\nabla} \times \boldsymbol{E}_{\text{FE-IIEE}}$ requires the evaluation of a double surface integral. The algorithm proposed for it is shown in Algorithm 1. It is designed to reuse the highest number of operations. The algorithm shows a loop unrolling which will also be used to feed the GPU efficiently, as will be discussed in Section III. Note that the computation of the electric and magnetic currents $\boldsymbol{J}_{\text{eq}}$, $\boldsymbol{M}_{\text{eq}}$, in lines 13 and 14, could also be realized in the inner loop in line 20. However, this is not efficient as it was to be repeated for every element in S.

III. GPU PARALLEL IMPLEMENTATION

A. CHALLENGES OF THE PARALLELIZATION

One of the main challenges of parallelizing the FE-IIEE code is to choose which of its loops is more appropriate to be parallelized using CUDA, as it will be clear later in Section III-B. The sequential version of the algorithm involves a large number of loops with five nesting levels (see Algorithm 1). Several of those loops could be fully parallelized launching a CUDA thread to execute each iteration. However, the complexity and granularity of the code on the loops are highly different and, in the same way, the data and the computations to be performed by each thread. Choosing one loop to be parallelized determines the data to transfer to the threads, the ratio between the computation and transference costs and also the information that can be shared among the threads. Even the innermost loop involves hundreds of floating point operations and tens of local data elements used by each thread. Besides, the number of iterations of each loop can be very different depending on the problem to solve, the discretization granularity (number of elements in the mesh) or the number of right-hand sides of the systems of equations involved. Therefore, the number of iterations defines the quantity of threads that can exploit the resources of the GPU and the granularity of the loop defines the workload of each of those threads. It is worth repeating here that the parallelization of the code is within the context of a minimum alteration of the original non-GPU code.

Algorithm 1 Computation of $\boldsymbol{E}_{\text{FE-IIEE}}$ and $\boldsymbol{\nabla} \times \boldsymbol{E}_{\text{FE-IIEE}}$ in All the Integration Points

Input: Elements from S and S'
Input: Number of RHS, n_{RHS}
1: ▷ Runs through all the elements on S to count the total number of integration points N_{int} needed to evaluate (8).
2: **for all** elem_S in S **do** ▷ Loop for S (size `numS`)
3: $n_{\text{int}} \leftarrow$ get_num_integration_points (elem_S)
4: $N_{\text{int}} \leftarrow N_{\text{int}} + n_{\text{int}}$ ▷ Accumulate for N_{int}
5: **end for** ▷ End loop for S
6: ▷Allocate for array `scat_vec`=($\boldsymbol{E}_{\text{FE-IIEE}}$,$\boldsymbol{\nabla} \times \boldsymbol{E}_{\text{FE-IIEE}}$)
7: `scat_vec` $\leftarrow$ zeros $(6 \cdot N_{\text{int}} \cdot n_{\text{RHS}})$
8: **for all** $\text{elem}_{S'}$ in S' **do** ▷ Loop for S' (size `numSp`)
9: $\boldsymbol{r'}_{\text{int}} \leftarrow$ get_integration_points ($\text{elem}_{S'}$)
10: **for all** $\boldsymbol{r'}_{\text{int}}$ **do** ▷ Loop for int. points on S' (size n'_{int})
11: ▷ Compute $\boldsymbol{J}_{\text{eq}}(\boldsymbol{r'})$ and $\boldsymbol{M}_{\text{eq}}(\boldsymbol{r'})$ through (13).
12: **for all** RHS **do** ▷ Loop for RHS (size n_{RHS})
13: $\boldsymbol{J}_{\text{eq}}(\boldsymbol{r'}_{\text{int}}) \leftarrow$ get_electric_current ($\boldsymbol{r'}_{\text{int}}$, RHS)
14: $\boldsymbol{M}_{\text{eq}}(\boldsymbol{r'}_{\text{int}}) \leftarrow$ get_magnetic_current ($\boldsymbol{r'}_{\text{int}}$, RHS)
15: **end for** ▷ End loop for RHS
16: **end for** ▷ End loop for int. points on S'
17: **for all** elem_S in S **do** ▷ Loop for S (size `numS`)
18: $\boldsymbol{r}_{\text{int}} \leftarrow$ get_integration_points (elem_S)
19: **for all** $\boldsymbol{r}_{\text{int}}$ **do** ▷ Loop int. points on S (size n_{int})
20: **for all** $\boldsymbol{r'}_{\text{int}}$ **do** ▷ Loop int. points on S' (size n'_{int})
21: ▷ Computes Green's function through (12).
22: $G(\boldsymbol{r}_{\text{int}}, \boldsymbol{r'}_{\text{int}}) \leftarrow$ get_greens_function($\boldsymbol{r}_{\text{int}}, \boldsymbol{r'}_{\text{int}}$)
23: **for all** RHS **do** ▷ Loop for RHS (size n_{RHS})
24: ▷ Scattered field and its curl through (10), (11)
25: $\text{u}_{\boldsymbol{E}} \leftarrow$ get_field($\boldsymbol{J}_{\text{eq}}$,$\boldsymbol{M}_{\text{eq}}$, G, RHS) ▷ (10)
26: $\text{u}_{\boldsymbol{\nabla} \times \boldsymbol{E}} \leftarrow$ get_curl($\boldsymbol{J}_{\text{eq}}$,$\boldsymbol{M}_{\text{eq}}$, G, RHS) ▷ (11)
27: ▷ Accumulate into `scat_vec`
28: `scat_vec` $\leftarrow$ `scat_vec` + $(\text{u}_{\boldsymbol{E}}, \text{u}_{\boldsymbol{\nabla} \times \boldsymbol{E}})$
29: **end for** ▷ End loop for RHS
30: **end for** ▷ End loop for int. points on S'
31: **end for** ▷ End loop for int. points on S
32: **end for** ▷ End loop for S
33: **end for** ▷ End loop for S'

In [53], we used CUDA to parallelize the innermost loop and obtained excellent performance using a few thousands of right-hand sides. However, most of the problems that can be solved using the FE-IIEE code involve only one or at most a few tens of right-hand sides. Therefore, in this paper we have chosen to parallelize another loop with a coarser level of granularity. Our experimental results show that the performance of the algorithm depends on the interaction among many factors that must be adjusted to the problem and to the architecture of the GPU. Those factors include the total number of threads, the thread-block size, the shared memory available to each block and the number of registers available to every thread.

B. GPU PARALLELIZATION

Algorithm 1 summarizes the mains steps of the IIEE part of the FE-IIEE method (in which the field $\boldsymbol{E}_{\text{FE-IIEE}}$, and its curl $\boldsymbol{\nabla} \times \boldsymbol{E}_{\text{FE-IIEE}}$ are computed) as five nested loops. It is worth noting that, once $\boldsymbol{E}_{\text{FE-IIEE}}$ and $\boldsymbol{\nabla} \times \boldsymbol{E}_{\text{FE-IIEE}}$ are obtained, they are combined following (14) in order to obtain the value of function $\boldsymbol{\Psi}$. This function is further integrated through $l_{\boldsymbol{\Psi}}(\boldsymbol{F})$ in expression (8), which is included in the right-hand side of the FEM algebraic system of equations.

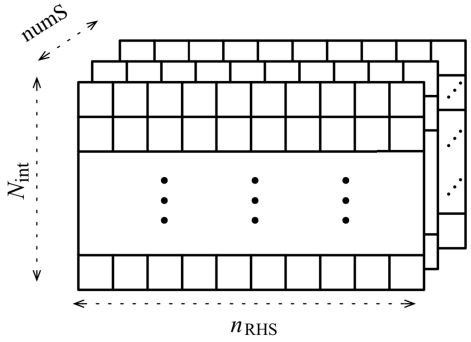

FIGURE 3. Logical structure and size of `scat_vec` computed by the algorithm.

We point out that a hybrid parallel methodology was originally present in the FEM code (including FE-IIEE part) using MPI processes and OpenMP threads within each process [51]. However there were still operations in the inner loops that were not parallelized and that were more suitable for execution on a GPU accelerator, since these loops involve massive regular computations with medium- or small-size granularity.

Fig. 3 depicts a 3D version of the `scat_vec`, a label used for denoting the complex scattering vector, computed by the FE-IIEE method and allocated in line 7 of Algorithm 1. This vector contains the values of $\boldsymbol{E}_{\text{FE-IIEE}}$ and $\boldsymbol{\nabla} \times \boldsymbol{E}_{\text{FE-IIEE}}$ (expressions (10), (11)) in the corresponding integration points. The first dimension corresponds to the number of right-hand sides (n_{RHS}), the second dimension to the total number of integration points on S (N_{int}), and the third dimension to the number of elements on S (`numS`). The elements are stored sequentially in memory, but we adopt a 3D representation as we can parallelize the update of the elements on different dimensions.

Each small square in Fig. 3 represents 6 complex elements of the vector modified by one iteration of the innermost loop (line 23) corresponding to a single right-hand side of the problem to solve. Every iteration of that loop involves dozens of products and additions of complex and real numbers that affect to different elements of `scat_vec`, and can be implemented as a kernel to be run by different threads on a GPU platform, [53]. In this work, the loop to be exploited by the many-core architecture of the GPU platforms is the one for S (line 17), in order to increase the number of applications where this acceleration can be leveraged (even for only one RHS, e.g., a single antenna). This loop corresponds to one layer in Fig. 3 and consists of thousands of iterations. To this end, we have implemented a CUDA kernel where each thread

is in charge of one iteration of that loop. Threads can perform their computations in lockstep modifying synchronously different elements of `scat_vec`. The kernel is launched once per iteration of the loop for S' (line 8) and receives as inputs the vectors $\boldsymbol{J}_{\text{eq}}$, $\boldsymbol{M}_{\text{eq}}$ shown in Algorithm 1 and also a vector containing the coordinates of the integration points on S. Two important issues to take into account to obtain an efficient implementation of the kernel are: 1) the occupancy of the GPU cores, and 2) the management of the different kinds of GPU memories, [54, Chap. 9]. In order to address the first issue, we tried to optimize the thread block size to leverage all the cores of the architecture.

As for the memory usage, we store the vector containing the result of computing the Green's function (line 22) and other auxiliary vectors computed in the innermost loop (line 23) into the local memory of the CUDA threads. Besides, at every iteration of the loop for RHS, only one thread per block copies the values of vectors $\boldsymbol{J}_{\text{eq}}$ and $\boldsymbol{M}_{\text{eq}}$ to the shared memory of the block. Using this technique, the number of accesses to global memory is reduced and this kind of fast memory is leveraged. Moreover, we minimized the occupancy of the static shared memory by copying only the 6 complex elements (96 bytes) required at each iteration of the loop. For example, these vectors represent a very small percentage of the 48 KB of this kind of memory available to each thread block on NVIDIA's Volta GPU architecture. Therefore, our use of this small shared memory is independent of the size of the problem.

In the most recent GPU architectures, the shared memory and L1 cache are combined into a single data cache. As we are using only 96 bytes of static shared memory, we configured the kernel to reduce the portion devoted to this kind of memory, know as *carveout*. However, there is no significant improvement on performance derived from this strategy.

Nevertheless, our analysis of the parallelization shows that the main factor that limits the performance of our CUDA code is the number of registers available in each streaming multiprocessor. Parallelizing the loop in line 17 of Algorithm 1 to enlarge the level of parallelism greatly increases the granularity and complexity of the computations of the kernel and so the number of local variables used in each thread. Therefore, the maximum occupancy of the GPU is not reached due to the number of registers per SM. There is a trade-off between the number of registers and the memory usage and we have used the best option to implement our code. Recall that a Volta architecture has 64K registers per SM, but it can have up to 64 warps active per multiprocessor.

IV. EXPERIMENTAL EVALUATION

We show results corresponding to two scenarios: (1) a radiation problem from a single antenna and (2) the scattering problem of the computation of the radar cross section (RCS) of an object. Regarding the antenna problem, $\boldsymbol{\Psi} = \mathbf{0}$ and there is only one RHS (the excitation of the antenna). On the other hand, for the RCS problem, $\boldsymbol{\Psi} \neq \mathbf{0}$ and multiple RHS (one for each desired direction of incidence) are present. All the computations are evaluated in double-precision, in contrast to, e.g., [34], [51], where single-precision was used.

In terms of hardware, performance tests were carried out in a server containing an Intel Xeon Gold 6126 processor, at 2.60 GHz, with 24 cores and 62 GB of DDR4-2666 main memory. This server also includes a state-of-the-art Tesla V100 PCIe GPU powered by the NVIDIA Volta architecture [55]. This GPU is composed of 80 Streaming Multiprocesor (SM) units with 64 CUDA cores per unit, i.e. 5,120 CUDA cores in total, a register file with 20,480 KB, an L2 cache with 6,144 KB, and a global memory of 32 GB.

The original FEM code was developed in Fortran 2003 [51]. Thus, a C wrapper was required in order to launch the CUDA kernel. The experiments were performed with Linux Ubuntu 19.04. The code was compiled with the 2019 version of the Intel Fortran, C compilers, and the CUDA verson 10.1. A customization layer over GiD is used to obtain FEM meshes and to plot the results, [56].

In order to leverage the persistence of the GPU memory among successive calls from Fortran to the C kernel, we use static memory to allocate the vectors to store at the global GPU memory in all the calls to the kernel. previous to the first iteration of the outermost loop, deallocating the memory at the end of that loop. Then, we update the same allocated vector `scat_vec` among successive calls and only retrieve the final result from the GPU to the CPU after the last iteration of the outermost loop.

A. ANTENNA PROBLEM

A pyramidal horn antenna as defined in [57, Chap. 9.4.3] is considered. The geometry of the problem is included in Fig. 4(a), and the specific dimensions are detailed in Table 1.

Six different meshes are tested for the same problem, with a decreasing size of the elements leading to meshes with the parameters shown in Table 2. The working frequency is set to 8.75 GHz. Near-field results inside the horn together with a directivity diagram are included in Fig. 5. Note that the expected directivity of this antenna is 21.8 dB [57]. Thus, an excellent agreement is obtained.

We have chosen to carry out a classic study of the parallel performance of the algorithm by analyzing its speedup with respect to its sequential version [58, Chap. 5]. This is a common practice in the literature, e.g., [29], [30], [33], [34]. This analysis assesses the quality of the CUDA parallelization on a GPU and the performance obtained from a manycore architecture. An alternative choice would be to compute the speedup with respect to a multi-threaded version of the code running only in the CPU. However, this latter option might have been misleading since the introduction of the multi-threaded part is not straightforward and relies on some design

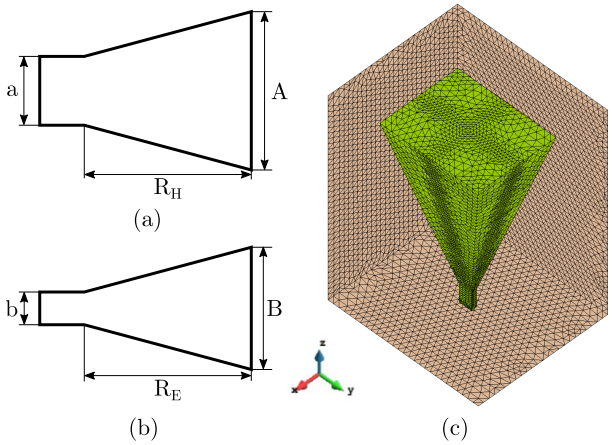

FIGURE 4. Definition of the pyramidal horn antenna. In (a) and (b), the cross section through H-plane and E-planes are shown, respectively, and in (c), the 3D isometric view (including the mesh truncation box) is pictured.

TABLE 1. Dimensions of the pyramidal horn antenna.

Parameter	*Value (mm)*
a	22.86
b	10.16
A	186.1
B	147.5
R_H	295.3
R_E	295.3

TABLE 2. Sequential times for the antenna problem.

Tetrahedra	numS	*Time (sec.)*
34,485	3,024	10.45
43,128	5,536	19.26
68,109	12,516	43.36
107,664	22,284	76.49
226,524	50,060	162.77
352,403	78,640	255.98

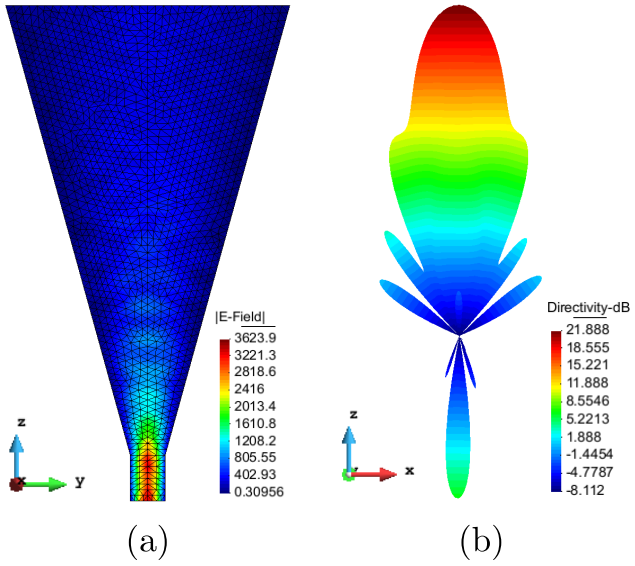

FIGURE 5. Results of the antenna simulation at $f = 8.75$ GHz: (a) Magnitude of the electric field, (b) Directivity.

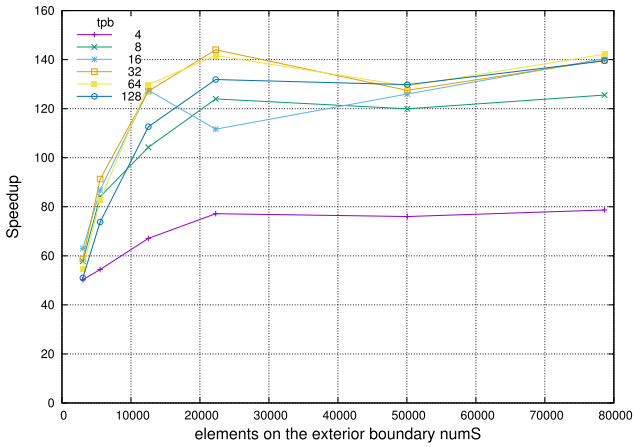

FIGURE 6. Effect of the thread-block size in the speedup of the GPU parallel algorithm with respect to one core of the CPU. Results with one right-hand side ($n_{\mathbf{RHS}} = 1$).

choices for the code that should be taken by the authors. Therefore, we evaluate the performance of the presented GPU parallel implementation of FE-IIEE with respect to its sequential execution in a modern CPU. The number of interior boundary elements of the surface S' (numSp) is, in all cases, 946 and thus, this is the number of times the kernel is launched. Table 2 shows the execution time of the sequential method on the CPU, with one RHS and using meshes with increasing number of exterior boundary elements. As it is shown, the sequential time grows linearly with the number of elements on S, numS.

Leveraging the many-core architecture of the GPU delivers very high speedups with respect to the sequential CPU execution. We have tested the CUDA code with different thread block sizes from 4 to 128 threads-per-block (tpb). Fig. 6 shows that the speedup usually increases with the value of tpb. Also, the speedup remains stable with the size of the problem once a minimum size (computational load) is reached, which is a very desirable behaviour which is not always achieved (see, e.g., [34]).

Fig. 7 shows the results obtained with the best thread block size on each case. It shows that the speedup grows with numS and reaches a value of 147 for the largest problem. As we increase numS, we have more difficulties to leverage the fastest memories of the GPU.

We have also assessed the effect of using the shared memory of the GPU on the performance of the algorithm. Fig. 8 compares the speedup obtained with two versions of our parallel algorithm using the best thread-block size on each case. On the first version, every block of threads uses static

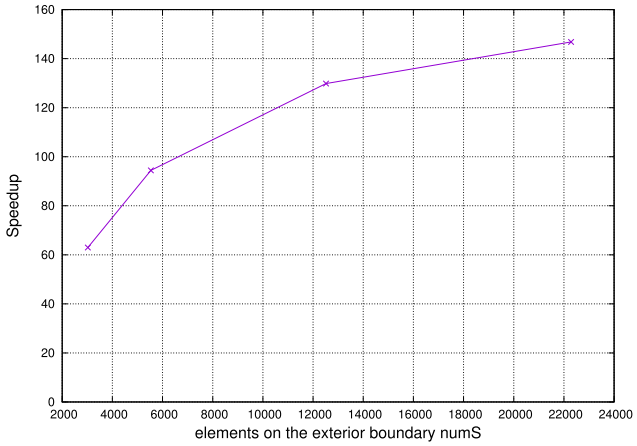

FIGURE 7. Speedup of the GPU parallel algorithm applied to the antenna problem with respect to one core of the CPU.

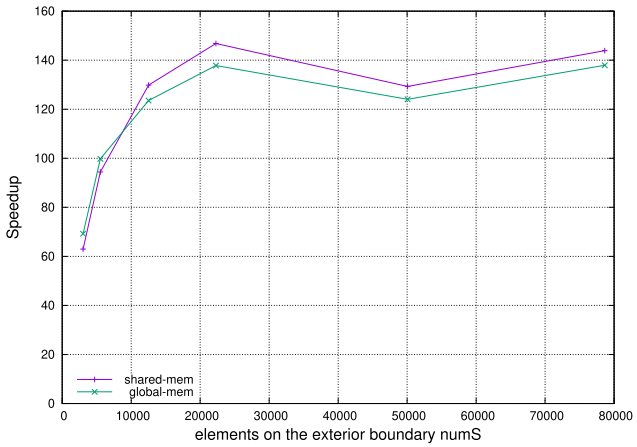

FIGURE 8. Effect of using shared memory on the speedup. Results with 1 right-hand side ($n_{\mathbf{RHS}} = 1$).

shared memory to store the values of vectors $\boldsymbol{J}_{eq}$ and $\boldsymbol{M}_{eq}$ as described at the beginning of this section. On the second version, every thread accesses those values on the much slower global memory of the GPU. We can see that, except in the case of the smaller problem, the use of shared memory produces a slight increment of the speedup of the algorithm.

B. RCS PROBLEM

We next address one of the RCS benchmark problems introduced in [59] (specifically, the ogive). A 3D view of the geometry is included in Fig. 9. The working frequency is set to 1.19 GHz. The vector function $\boldsymbol{\Psi}_{\text{inc}}$ is computed through proper substitution of the mathematical expression corresponding to a plane wave in (4). Multiple RHS are mandatory in the case of computing monostatic RCS for a sweep of spherical angles. The bistatic RCS for an incident wave at $\theta = 90°$, directed to the tip of the ogive in vertical polarization, is also shown in Fig. 9. A number of five meshes, starting from 27,739 tetrahedra and introducing increasing levels of refinement, have been simulated.

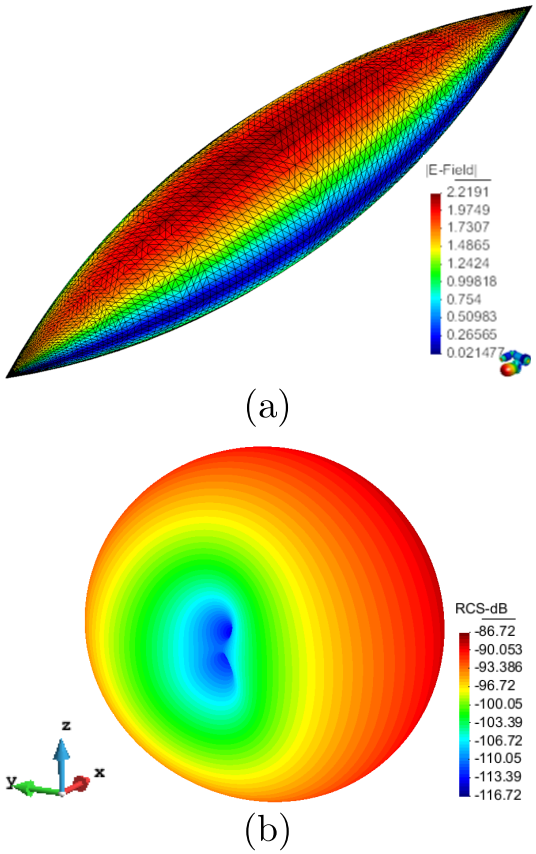

FIGURE 9. Ogive simulated as RCS problem: (a) Mesh, (b) Bistatic RCS (in dbλ^2 units).

TABLE 3. Sequential times for the RCS problem.

numS	n_{RHS} 1	10
1,572	1.66	9.63
3,432	3.61	21.08
8,012	8.41	49.14
14,696	15.43	90.19
32,576	34.26	200.12

Table 3 shows the time corresponding to the sequential execution with one and ten RHS and using meshes with increasing number of exterior boundary elements numS. The number of interior boundary elements numSp is always 300. We can observe that the sequential time grows linearly not only with numS, but also with the number of RHS.

This problem allows us to analyze the effect of the number of RHS on the optimal thread block size. In the case of one RHS, the best results were obtained using 32 or 64 threads per block. On the contrary, in the case of 10 RHS the best results were obtained with 8 or 16 threads per block, which is smaller than the warp size for the experimental platform. Launching less threads per block when the problem size grows is a good strategy, as it increases the number or registers and the shared memory available to each thread.

Fig. 10 shows the effect of increasing both numS and n_{RHS} on the speedup of the CUDA implementation. It illustrates

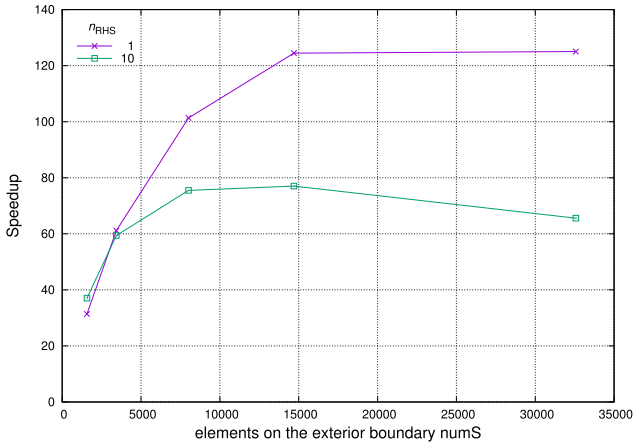

FIGURE 10. Speedup of the GPU parallel algorithm applied to the RCS problem with respect to one core of the CPU.

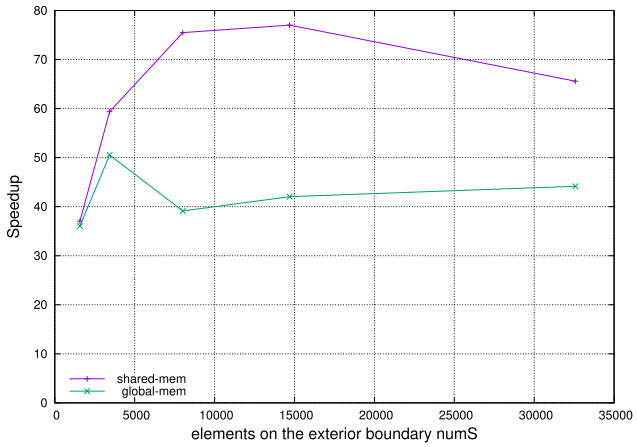

FIGURE 11. Effect of using shared memory on the speedup. Results with 10 right-hand sides ($n_{RHS} = 10$).

that the speedup grows with numS. In the case of 10 RHS it only decreases with the largest numS because the vectors do not fit on the faster memories of the GPU and the number of transactions with global and L2 cache memories quickly increases with the size of the problem.

The results also show that the best results are obtained with one RHS due to the GPU memory access pattern. As we can observe in Fig. 3, for a single RHS every thread accesses memory positions much closer to each other that when we increase the number of RHS. This allows a more coalesced access to memory, which is an important factor to obtain high performance on GPUs, [54, Chap. 9].

We have also assessed the effect of using shared memory to solve this problem. Unlike in the case of the antenna problem (see Fig. 11) we can clearly see the large impact of this optimization on the speedup when using shared memory to solve the RCS problem. Only with small problems both versions of the algorithm obtain similar speedups on this particular problem and configuration. Thus, we have been able to verify that the behaviour of this CUDA optimization applied to the FE-IIEE code depends not only on the computational

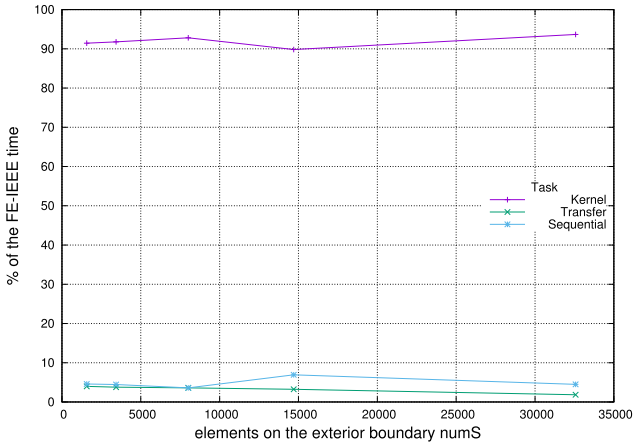

FIGURE 12. Percentage of the FE-IIEE execution time devoted to different tasks with 10 right-hand sides.

granularity of the parallel kernel given by the number of right-hand sides (n_{RHS}), but also on the discretization granularity of the domains of the problem (numSp and numS). Besides, this behaviour could be quite different depending on the resources provided by the GPU architecture.

Finally, the negative effect of the transfer time between the CPU and the GPU or the time of the sequential component of the code is very small, as shown in Fig. 12. They both decrease from 12% to 5% of the total time with one RHS and they always represent less than 5% with 10 RHS. Thus, we conclude that the presented implementation limits the negative effect of transferring information between the CPU and the GPU even when increasing the size of the problem.

V. CONCLUSIONS

In this paper, we have shown that the efficient use of CUDA on a state-of-the-art GPU allows us to obtain very large speedups in the solution of open region electromagnetic problems. Specifically, a non-standard FEM mesh truncation technique (called FE-IIEE) has been successfully implemented in CUDA. Most of the performance of our implementation arises from leveraging the computational power of the thousands of cores of the GPU. We increased the granularity of the computations on each thread in order to reduce the number of kernel launches and the effect of transferring data between CPU and GPU. Besides, we reduced the use of global memory by storing most of the data on the faster local and shared memories available to the threads. We have also demonstrated that the best thread-block size is not always a multiple of the warp size. Indeed, this value depends on the size of the problem and the local and shared memory available to each thread.

Without loss of generality, two specific electromagnetic problems have been solved: a pyramidal horn as antenna problem and the computation of the RCS for an ogive. Speedups higher than 140 on a Volta GPU have been obtained thanks to the performance of our non-intrusive GPU parallel algorithm.

As future work, we intend to implement new versions of the algorithm using OpenMP to parallelize the loop for S (line 17 of Algorithm 1) and compare it with the CUDA version introduced on this paper and the OpenMP version introduced by two of the authors on [51]. We also intend to implement a multi-GPU version of the algorithm trying to improve its scalability even more.

REFERENCES

[1] P. Monk, *Finite Element Methods for Maxwell Equations*. London, U.K.: Oxford Univ. Press, 2003.

[2] M. Salazar-Palma, T. K. Sarkar, L. E. García-Castillo, T. Roy, and A. R. Djordjevic, *Iterative and Self-Adaptive Finite-Elements in Electromagnetic Modeling*. Norwood, MA, USA: Artech House, 1998.

[3] J. M. Jin, *The Finite Element Method in Electromagnetics*, 2nd ed. Hoboken, NJ, USA: Wiley, 2002.

[4] P. G. Ciarlet, *The Finite Element Methods for Elliptic Problems*. New York, NY, USA: North Holland, 1994.

[5] J. M. Jin and D. J. Ryley, *Finite Element Analysis of Antennas and Arrays*. Hoboken, NJ, USA: Wiley, 2009.

[6] P.-H. Tournier, I. Aliferis, M. Bonazzoli, M. D. Buhan, M. Darbas, V. Dolean, F. Hecht, P. Jolivet, I. El Kanfoud, C. Migliaccio, F. Nataf, C. Pichot, and S. Semenov, "Microwave tomographic imaging of cerebrovascular accidents by using high-performance computing," *Parallel Comput.*, vol. 85, pp. 88–97, Jul. 2019.

[7] E. S. Um, J. Kim, M. J. Wilt, M. Commer, and S.-S. Kim, "Finite-element analysis of top-casing electric source method for imaging hydraulically active fracture zones," *Geophysics*, vol. 84, no. 1, pp. E23–E35, 2019.

[8] S. M. Musa, *Computational Finite Element Methods in Nanotechnology*. Boca Raton, FL, USA: CRC Press, 2012.

[9] T. B. A. Senior and J. L. Volakis, *Approximate Boundary Conditions in Electromagnetics*. London, U.K.: IET, 1995.

[10] J.-P. Berenger, "A perfectly matched layer for the absorption of electromagnetic waves," *J. Comput. Phys.*, vol. 114, no. 2, pp. 185–200, Oct. 1994.

[11] P. Bettess, "Infinite elements," *Int. J. Numer. Methods Eng.*, vol. 11, no. 1, pp. 54–64, 1977.

[12] I. Gómez-Revuelto, L. E. García-Castillo, and L. F. Demkowicz, "A comparison between PML, infinite elements and an iterative BEM as mesh truncation methods for *hp* self-adaptive procedures in electromagnetics," *Prog. Electromagn. Res.*, vol. 126, pp. 499–519, 2012. [Online]. Available: http://www.jpier.org/PIER/pier.php?volume=126

[13] P. Silvester and M.-S. Hsieh, "Finite-element solution of 2-dimensional exterior-field problems," *Proc. Inst. Electr. Eng.*, vol. 118, no. 12, pp. 1743–1747, Dec. 1971.

[14] L. E. García-Castillo, I. Gómez-Revuelto, F. S. de Adana, and M. Salazar-Palma, "A finite element method for the analysis of radiation and scattering of electromagnetic waves on complex environments," *Comput. Methods Appl. Mech. Eng.*, vol. 194, nos. 2–5, pp. 637–655, Feb. 2005.

[15] I. Gómez-Revuelto, L. E. García-Castillo, M. Salazar-Palma, and T. K. Sarkar, "Fully coupled hybrid-method FEM/high-frequency technique for the analysis of 3D scattering and radiation problems," *Microw. Opt. Technol. Lett.*, vol. 47, no. 2, pp. 104–107, Oct. 2005.

[16] R. Fernández-Recio, L. E. García-Castillo, I. Gómez-Revuelto, and M. Salazar-Palma, "Fully coupled hybrid FEM-UTD method using NURBS for the analysis of radiation problems," *IEEE Trans. Antennas Propag.*, vol. 56, no. 3, pp. 774–783, Mar. 2008.

[17] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: A pedestrian prescription," *IEEE Antennas Propag. Mag.*, vol. 35, no. 3, pp. 7–12, Jun. 1993.

[18] N. A. Gumerov and R. Duraiswami, *Fast Multipole Methods for Helmholtz Equation Three Dimensions* (Elsevier Series in Electromagnetism), I. Mayergoyz, Ed. Amsterdam, The Netherlands: Elsevier, 2004.

[19] J. R. Phillips and J. K. White, "A precorrected-FFT method for electrostatic analysis of complicated 3-D structures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 10, pp. 1059–1072, Oct. 1997.

[20] J. R. Phillips, "Error and complexity analysis for a collocation-grid-projection plus precorrected-FFT algorithm for solving potential integral equations with Laplace or Helmholtz kernels," in *Proc. Copper Mountain Conf. Multigrid Methods*, 1995, pp. 673–688.

[21] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "AIM: Adaptive integral method for solving large-scale electromagnetic scattering and radiation problems," *Radio Sci.*, vol. 31, no. 5, pp. 1225–1251, Sep. 1996.

[22] S. Mo Seo and J.-F. Lee, "A fast IE-FFT algorithm for solving PEC scattering problems," *IEEE Trans. Magn.*, vol. 41, no. 5, pp. 1476–1479, May 2005.

[23] S. M. Seo and J.-F. Lee, "A single-level low rank IE-QR algorithm for PEC scattering problems using EFIE formulation," *IEEE Trans. Antennas Propag.*, vol. 52, no. 8, pp. 2141–2146, Aug. 2004.

[24] M. Bebendorf, "Approximation of boundary element matrices," *Numerische Math.*, vol. 86, no. 4, pp. 565–589, Oct. 2000.

[25] R. M. Barrio-Garrido, L. E. García-Castillo, I. Gómez-Revuelto, and M. Salazar-Palma, "Self-adaptive hp finite element method with iterative mesh truncation technique accelerated with adaptive cross approximation," *Comput. Math. Appl.*, vol. 71, no. 10, pp. 1911–1932, May 2016.

[26] Z. Fu, T. J. Lewis, R. M. Kirby, and R. T. Whitaker, "Architecting the finite element method pipeline for the GPU," *J. Comput. Appl. Math.*, vol. 257, pp. 195–211, Feb. 2014.

[27] K. Świrydowicz, N. Chalmers, A. Karakus, and T. Warburton, "Acceleration of tensor-product operations for high-order finite element methods," *Int. J. High Perform. Comput. Appl.*, vol. 33, no. 4, Jul. 2019, Art. no. 1094342018816368.

[28] A. Karakus, N. Chalmers, K. Świrydowicz, and T. Warburton, "A GPU accelerated discontinuous Galerkin incompressible flow solver," *J. Comput. Phys.*, vol. 390, pp. 380–404, Aug. 2019.

[29] Y. Wang, Q. Wang, X. Deng, Z. Xia, J. Yan, and H. Xu, "Graphics processing unit (GPU) accelerated fast multipole BEM with level-skip M2L for 3D elasticity problems," *Adv. Eng. Softw.*, vol. 82, pp. 105–118, Apr. 2015.

[30] S. Peng and Z. Nie, "Acceleration of the method of moments calculations by using graphics processing units," *IEEE Trans. Antennas Propag.*, vol. 56, no. 7, pp. 2130–2133, Jul. 2008.

[31] J. Guan, S. Yan, and J.-M. Jin, "An OpenMP-CUDA implementation of multilevel fast multipole algorithm for electromagnetic simulation on multi-GPU computing systems," *IEEE Trans. Antennas Propag.*, vol. 61, no. 7, pp. 3607–3616, Jul. 2013.

[32] V. Dang, Q. M. Nguyen, and O. Kilic, "GPU cluster implementation of FMM-FFT for large-scale electromagnetic problems," *IEEE Antennas Wireless Propag. Lett.*, vol. 13, pp. 1259–1262, 2014.

[33] Z. Lin, Y. Chen, Y. Zhang, X. Zhao, and H. Zhang, "An efficient GPU-based Out-of-Core LU solver of parallel higher-order method of moments for solving airborne array problems," *Int. J. Antennas Propag.*, vol. 2017, pp. 1–10, 2017.

[34] E. García, C. Delgado, L. Lozano, and F. Cátedra, "Efficient strategy for parallelisation of multilevel fast multipole algorithm using CUDA," *IET Microw., Antennas Propag.*, vol. 13, no. 10, pp. 1554–1563, Aug. 2019.

[35] H.-T. Meng, B.-L. Nie, S. Wong, C. Macon, and J.-M. Jin, "GPU accelerated finite-element computation for electromagnetic analysis," *IEEE Antennas Propag. Mag.*, vol. 56, no. 2, pp. 39–62, Apr. 2014.

[36] A. Dziekonski, P. Sypek, A. Lamecki, and M. Mrozowski, "Finite element matrix generation on a GPU," *Prog. Electromagn. Res.*, vol. 128, pp. 249–265, 2012. [Online]. Available: http://www.jpier.org/PIER/pier.php?volume=128

[37] A. Dziekonski, P. Sypek, A. Lamecki, and M. Mrozowski, "Communication and load balancing optimization for finite element electromagnetic simulations using multi-GPU workstation," *IEEE Trans. Microw. Theory Techn.*, vol. 65, no. 8, pp. 2661–2671, Aug. 2017.

[38] A. Dziekonski and M. Mrozowski, "A GPU solver for sparse generalized eigenvalue problems with symmetric complex-valued matrices obtained using higher-order FEM," *IEEE Access*, vol. 6, pp. 69826–69834, 2018.

[39] J. Guan, S. Yan, and J.-M. Jin, "An accurate and efficient finite element-boundary integral method with GPU acceleration for 3-D electromagnetic analysis," *IEEE Trans. Antennas Propag.*, vol. 62, no. 12, pp. 6325–6336, Dec. 2014.

[40] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[41] J. A. Belloch, A. Gonzalez, A. M. Vidal, and M. Cobos, "On the performance of multi-GPU-based expert systems for acoustic localization involving massive microphone arrays," *Expert Syst. Appl.*, vol. 42, no. 13, pp. 5607–5620, Aug. 2015.

[42] J. A. Belloch, A. Gonzalez, E. S. Quintana-Ortí, M. Ferrer, and V. Välimäki, "GPU-based dynamic wave field synthesis using fractional delay filters and room compensation," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 25, no. 2, pp. 435–447, Feb. 2017.

[43] *CUDA C Programming Guide*, document PG-02829-001_v1, Nvidia, Aug. 2019.
[44] L. E. García-Castillo and M. Salazar-Palma, "Second-order Nédélec tetrahedral element for computational electromagnetics," *Int. J. Numer. Model. Electron. Netw., Devices Fields*, vol. 13, nos. 2–3, pp. 261–287, Mar./Jun. 2000.
[45] R. F. Harrington, *Time Harmonic Electromagnetic Fields*. New York, NY, USA: McGraw-Hill, 1961.
[46] R. Fernández-Recio, L. E. García-Castillo, S. L. Romano, and I. Gómez-Revuelto, "Convergence study of a non-standard Schwarz domain decomposition method for finite element mesh truncation in electromagnetics," *Prog. Electromagn. Res.*, vol. 120, pp. 439–457, 2011. [Online]. Available: http://www.jpier.org/PIER/pier.php?volume=120
[47] I. Gómez-Revuelto, L. E. García-Castillo, and M. Salazar-Palma, "Goal-oriented self-adaptive *Hp*-strategies for scattering and radiation problems," *Prog. Electromagn. Res.*, vol. 125, pp. 459–482, 2012. [Online]. Available: http://www.jpier.org/PIER/pier.php?volume=125
[48] D. Garcia-Donoro, I. Martinez-Fernandez, L. E. García-Castillo, Y. Zhang, and T. K. Sarkar, "RCS computation using a parallel in-core and out-of-core direct solver," *Prog. Electromagn. Res.*, vol. 118, pp. 505–525, 2011. [Online]. Available: http://www.jpier.org/PIER/pier.php?volume=118
[49] R. M. Barrio-Garrido, L. E. García-Castillo, I. Gómez-Revuelto, and M. Salazar-Palma, "Self-adaptive *Hp* finite element method with iterative mesh truncation technique accelerated with adaptive cross approximation," *Comput. Math. Appl.*, vol. 71, no. 10, pp. 1911–1932, May 2016.
[50] D. Garcia-Donoro, I. Martinez-Fernandez, L. E. García-Castillo, and M. Salazar-Palma, "HOFEM: Higher order finite element method simulator for antenna analysis," in *Proc. Int. Conf. Antenna Meas. Appl. Focus Antenna Syst. (CAMA)*, New York, NY, USA, Oct. 2016, pp. 1–4.
[51] A. Amor-Martin, D. Garcia-Donoro, and L. E. García-Castillo, "Higher-order finite element electromagnetics code for HPC environments," in *Proc. Int. Conf. Comput. Sci. (ICCS)*, Zurich, Switzerland, Jun. 2017, pp. 818–827.
[52] D. Garcia-Donoro, S. Ting, L. E. García-Castillo, Y. Zhang, and T. K. Sarkar, "Higher order finite element method solver for RCS computation of complex targets," in *Proc. IET Int. Radar Conf.*, Hangzhou, China, Oct. 2015, pp. 1–4.
[53] J. A. Belloch, A. Amor-Martin, D. Garcia-Donoro, F. J. Martínez-Zaldívar, and L. E. García-Castillo, "On the use of many-core machines for the acceleration of a mesh truncation technique for FEM," *J. Supercomput.*, vol. 75, no. 3, pp. 1686–1696, Mar. 2019, doi: 10.1007/s11227-018-02739-9.
[54] *CUDA C Best Practices Guide*, document DG-05603-001_v1, Nvidia, Aug. 2019.
[55] *NVIDIA Tesla V100 GPU Architecture Whitepaper*, document WP-08608-001_v1, Aug. 2017.
[56] A. Melendo, A. Coll, M. Pasenau, E. Escolano, and A. Monros. Accessed: Nov. 2019. [Online]. Available: https://www.gidhome.com
[57] W. L. Stutzman and G. A. Thiele, *Antenna Theory and Design*. Hoboken, NJ, USA: Wiley, 2012.
[58] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing*, 2nd ed. Reading, MA, USA: Addison-Wesley, 2003.
[59] A. C. Woo, H. T. Wang, M. J. Schuh, and M. L. Sanders, "EM programmer's notebook-benchmark radar targets for the validation of computational electromagnetics programs," *IEEE Antennas Propag. Mag.*, vol. 35, no. 1, pp. 84–89, Feb. 1993.

JOSÉ M. BADÍA was born in Valencia, Spain. He received the B.S. degree in computer science from the Polytechnic University of Valencia, Spain, in 1991, and the Ph.D. degree in computer science in 1996.

Since 1994, he has been a member of the Department of Computer Science and Engineering, University Jaume I of Castellón, Spain, where he was a Teaching Assistant, from 1994 to 1997, an Assistant Professor, from 1997 to 2000, and an Associate Professor, since 2000. From 2007 to 2013, he was the Head of the Department. He has published more than 40 articles in international conferences and journals. His main research interests include high performance computing for dense and sparse algebra, power aware computing, and parallel data mining.

ADRIAN AMOR-MARTIN (Member, IEEE) was born in Móstoles, Madrid, in 1989. He received the degree in telecommunications engineering, the M.Sc. degree in multimedia and communications, and the Ph.D. degree in multimedia and communications from the University Carlos III of Madrid, in 2012, 2014, and 2018, respectively. He participated as a Researcher with the University Carlos III of Madrid, from 2012 to 2018 with different scholarships obtained on a competitive basis. Since 2019, he has been a Postdoctoral Researcher with the Universität des Saarlandes, Germany. He has authored eight publications in international journals and 16 contributions to international conferences. His research interests are focused on the application of numerical methods to high-performance computational electromagnetics including finite elements, domain decomposition methods, definition of basic functions, and hp adaptivity.

JOSE A. BELLOCH (Member, IEEE) received the degree in telecommunications engineering, the master's degree in parallel and distributed computing, and the Ph.D. degree in computer science from the Universitat Politècnica de València, Valencia, Spain, in 2007, 2010, and 2014, respectively. Since 2017, he has been an Assistant Professor with the Electronic Technology Department, Universidad Carlos III de Madrid, Madrid, Spain. He was a Visiting Researcher with the Department of Signal Processing and Acoustics, Aalto University School of Electrical Engineering, Espoo, Finland, as a Predoctoral Researcher, in 2013, and as a Postdoctoral Researcher, in 2015. He carried out an internship with the Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary. His current research interests include centered in applying the new parallel architectures into signal processing algorithms. He has developed several real-time audio applications related with multichannel massive filtering, binaural sound, wave field synthesis systems, and sound source localization using general purpose graphic processing units and ARM architectures. He was a recipient of the Extraordinary Ph.D. Thesis Award in recognition of his Ph.D. thesis. In 2016, he was honored with a Postdoctoral Fellowship of the Regional Government of Valencia in order to carry out research with the Universitat Jaume I de Castellón in collaboration with the Universidad Complutense de Madrid, Madrid.

LUIS EMILIO GARCÍA-CASTILLO (Member, IEEE) was born in Madrid, Spain, in 1967. He received the degree in ingeniero de telecomunicación and the Ph.D. degree from the Universidad Politécnica de Madrid, in 1992 and 1998, respectively. His Ph.D. thesis received two prizes from the Colegio Oficial de Ingenieros de Telecomunicación, Spain, and the Universidad Politécnica de Madrid.

From 1997 to 2000, he was an Associate Professor with the Universidad Politécnica de Madrid. From 2000 to 2005, he was an Associate Professor at the Universidad de Alcalá, Madrid. Since October 2005, he has been with the Department of Signal Theory and Communications, Universidad Carlos III de Madrid, Spain. His research activity and interests are focused in the application of numerical methods to high performance computational electromagnetics including finite elements, hp adaptivity, hybrid methods, and domain decomposition methods.

He has authored one book, five contributions for chapters and articles in books, 45 articles in international journals, and over 100 articles in international conferences, symposiums, and workshops, plus a number of national publications and reports. He has leaded as a Principal Investigator five projects of the National Plan of Research of Spain, one of the Regional Plan of Research of Madrid, and one with the American Air Force Office of Scientific Research. He has also participated in a number of projects and contracts, financed by international, European, and national institutions and companies.